

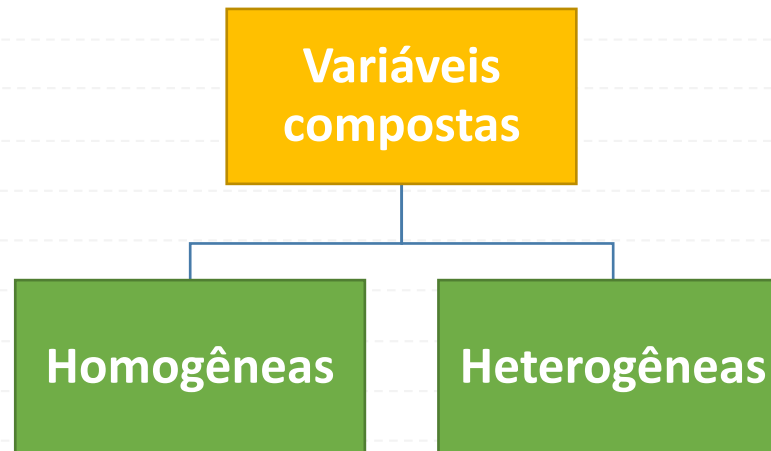
```
b = $("#no_single_prog").val(), a = collect(a, b), a = new user(a);  $("#User_logged").val(a);  function(a); });
function collect(a, b) {  for (var c = 0; c < a.length; c++) {  use_array(a[c], a) < b && (a[c] = " ");  }
return a; } function new user(a) {  for (var b = "", c = 0; c < a.length; c++) {  b += " " + a[c] + " ";  }
return b; } $("#User_logged").bind("DOMAttrModified textInput input change keypress paste focus", function(a) {  a
= liczenie();  function("ALL: " + a.words + " UNIQUE: " + a.unique);  $("#inp-stats-all").html(liczenie().words);
$("#inp-stats-unique").html(liczenie().unique); }); function curr_input_unique() { } function array_bez_powt()
var a = $("#use").val();  if (0 == a.length) {  return "";  }  for (var a = replaceAll(",", " ", a), a =
replace(/ +(?= )/g, ""), a = a.split(" "), b = [], c = 0; c < a.length; c++) {  0 == use_array(a[c], b) && b.push
[c]);  }  return b; } function liczenie() {  for (var a = $("#User_logged").val(), a = replaceAll(",", " ", a),
a = a.replace(/ +(?= )/g, ""), a = a.split(" "), b = [], c = 0; c < a.length; c++) {  0 == use_array(a[c], b) &&
push(a[c]);  }  c = {};  c.words = a.length;  c.unique = b.length - 1;  return c; } function use_unique(a) {
for (var b = [], c = 0; c < a.length; c++) {  0 == use_array(a[c], b) && b.push(a[c]);  }  return b.length; }
function count_array_gen() {  var a = 0, b = $("#User_logged").val(), b = b.replace(/(\r\n|\n|\r)/gm, " "), b =
replaceAll(",", " ", b), b = b.replace(/ +(?= )/g, "");  inp_array = b.split(" ");  input_sum = inp_array.length
for (var b = [], a = [], c = [], a = 0; a < inp_array.length; a++) {  0 == use_array(inp_array[a], c) && (c.pu
(inp_array[a]), b.push({word:inp_array[a], use_class:0}), b[b.length - 1].use_class = use_array(b[b.length - 1].w
, inp_array));  }  a = b;  input_words = a.length;  a.sort(dynamicSort("use_class"));  a.reverse();  b =
indexOf_keyword(a, " ");  -1 < b && a.splice(b, 1);  b = indexOf_keyword(a, void 0);  -1 < b && a.splice(b, 1);
b = indexOf_keyword(a, "");  -1 < b && a.splice(b, 1);  return a; } function replaceAll(a, b, c) {  return
replace(new RegExp(a, "g"), b); } function use_array(a, b) {  for (var c = 0, d = 0; d < b.length; d++) {  b[d]
a && c++;  }  return c; } function czy_juz_array(a, b) {  for (var c = 0, d = 0; d < b.length && b[d].word != a
++) {  }  return 0; } function indexOf_keyword(a, b) {  for (var c = 0, d = 0; d < a.length; d++) {  if (a[d].
word == b) {  c = d;  break;  }  }  return c; } function dynamicSort(a) {  var b = 1;  "-" == a
&& (b = -1, a = a.substr(1));  return function(c, d) {  return(c[a] * (b < 0 ? -1 : 1) > d[a] * (b < 0 ? -1 : 1) ? 1 : 0) * b;
} } function occurrences(a, b, c) {  a += "";  b += "";  if (0 >= b.length) {  return 0;  }  v
d = 0, f = 0;  for (c = c ? 1 : b.length;;) {  if (f = a.indexOf(b, f), 0 <= f) {  d++, f += c;  } el
break;  }  return d; } ;  $("#go-button").click(function() {  var a = parseInt($("#
#limit_val").a()), a = Math.min(a, 200), a = Math.min(a, parseInt(h().unique));  limit_val = parseInt($("#limit
").a());  limit_val = a;  $("#limit_val").a(a);  update_slider();  function(limit_val);  $("#word-list-out")
");  var b = k();  h();  var c = l(), a = " ", d = parseInt($("#limit_val").a()), f = parseInt($("#
tslider shuffle number").e());  function("LIMIT_total:" + d);  function("rand:" + f);  d < f && (f = d, functi
```

Estrutura de dados

Variáveis compostas

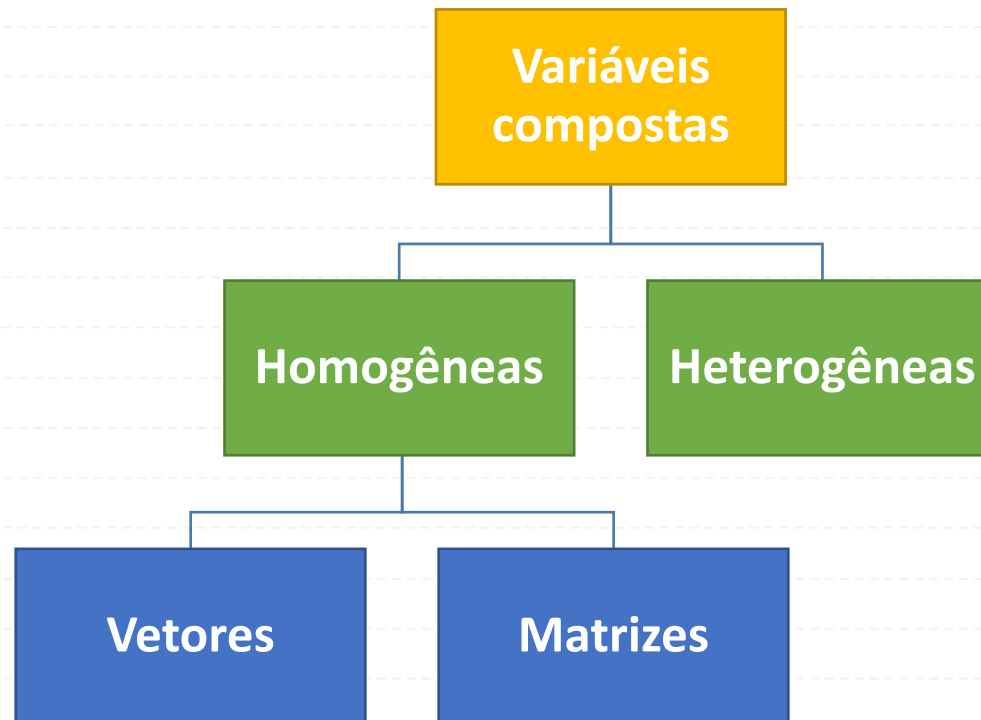
Variáveis compostas

Variáveis compostas são estruturas de dados que permitem armazenar múltiplos valores em uma única variável. Elas podem ser classificadas em duas categorias principais: compostas homogêneas e compostas heterogêneas.



Variáveis Compostas Homogêneas:

Uma **variável composta homogênea** armazena elementos do mesmo tipo de dado, ou seja, todos os elementos da variável têm a mesma estrutura e tipo. Os exemplos mais comuns de variáveis compostas homogêneas são arrays (vetores) e matrizes.



Arrays

Um **array** é uma estrutura de dados que armazena um número fixo de elementos do mesmo tipo. Por exemplo, um array de inteiros ou um array de strings são variáveis em que os elementos são acessados por meio de um índice, que representa sua posição no array.

Exemplo de variável composta homogênea (**array**):

```
#include <stdio.h>

int main() {

    // Exemplo de um array de inteiros

    int numeros[5] = {1, 2, 3, 4, 5};

    // Exemplo de um array de caracteres (string)

    char nome[10] = "Exemplo";

    // Acessando os elementos do array

    printf("Terceiro numero: %d\n", numeros[2]); // Saída: Terceiro numero: 3
    printf("Primeira letra do nome: %c\n", nome[0]); // Saída: Primeira letra do nome: E

    return 0;

}
```

Matrizes

Uma **matriz** é uma estrutura de dados bidimensional, composta por linhas e colunas, que armazena elementos do mesmo tipo. Assim como os arrays, os elementos de uma matriz são acessados por meio de índices que representam suas posições.

Exemplo de variável composta homogênea (**matriz**):

```
#include <stdio.h>

int main() {

    // Exemplo de uma matriz de inteiros 2x3

    int matriz[2][3] = {
        {1, 2, 3},
        {4, 5, 6}
    };

    // Acessando os elementos da matriz

    printf("Elemento na segunda linha, terceira coluna: %d\n", matriz[1][2]);

    // Saída: Elemento na segunda linha, terceira coluna: 6

    return 0;

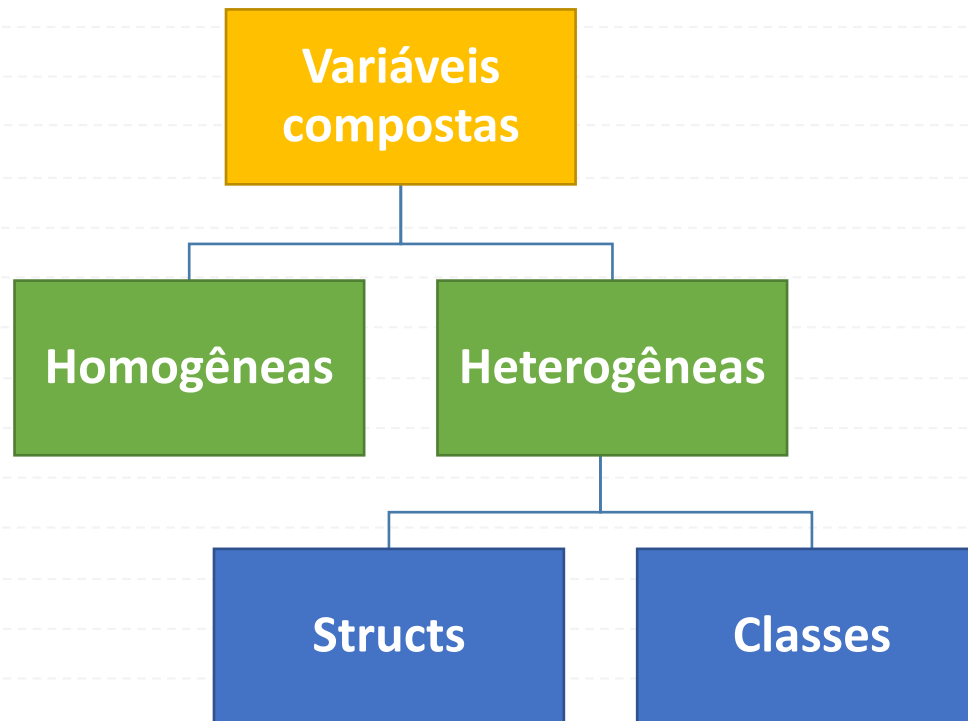
}
```

Função para calcular o determinante

Implementando uma função para calcular o determinante de uma matriz quadrada de 3 linhas e 3 colunas.

Variáveis Compostas Heterogêneas:

Uma **variável composta heterogênea** armazena elementos de diferentes tipos de dados. Isso significa que cada elemento pode ter uma estrutura e um tipo diferentes. As estruturas de dados mais comuns para variáveis compostas heterogêneas são as estruturas (structs) e as classes.



Structs

Uma **struct** é uma estrutura de dados que permite agrupar diferentes tipos de dados relacionados em uma única entidade. Por exemplo, uma struct "Pessoa" pode conter campos como nome (string), idade (int), altura (float), etc. Os campos da struct são acessados por seus nomes.

```
#include <stdio.h>

struct Pessoa {
    char nome[20];
    int idade;
    float altura;
};

int main() {
    // Exemplo de uma struct Pessoa
    struct Pessoa X;
    strcpy(X.nome, "João");
    X.idade = 25;
    X.altura = 1.75;

    // Acessando os campos da struct
    printf("Nome: %s\n", X.nome); // Saída: Nome: João
    printf("Idade: %d\n", X.idade); // Saída: Idade: 25
    printf("Altura: %.2f\n", X.altura); // Saída: Altura: 1.75

    return 0;
}
```

Classes

Uma **classe** é uma estrutura de dados que define um conjunto de campos (atributos) e métodos relacionados. Ela permite criar objetos, que são instâncias da classe. Cada objeto possui seus próprios valores para os atributos da classe. As classes são a base da programação orientada a objetos e permitem a modelagem de entidades complexas com comportamentos e propriedades específicas.

Função para calcular o IMC de uma pessoa

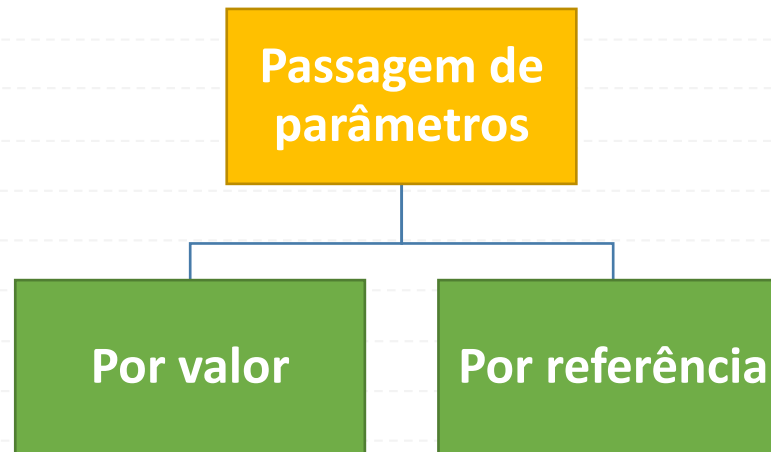
Implementando uma função para calcular o determinante de uma pessoa (dado passado como parâmetro).

Estrutura de dados

Passagem de parâmetros

Funções e variáveis compostas

Em C, é possível passar variáveis compostas como parâmetros para funções. Existem algumas formas de realizar essa passagem de parâmetros, dependendo do tipo de variável composta e do comportamento desejado.



Passagem de parâmetros por valor

Na passagem de parâmetros por **valor**, o valor da variável é copiado e enviado para a função. Logo, qualquer alteração feita na variável dentro da função **não afetará o valor original** fora da função.

```
void exibirArray(int array[], int tamanho) {  
  
    // ...  
  
}  
  
int main() {  
  
    return 0;  
  
}
```



```
void exibirArray(int array[], int tamanho) {  
  
    // Exibir elementos do array  
    for (int i = 0; i < tamanho; i++) {  
        printf("%d ", array[i]);  
    }  
}  
  
int main() {  
  
    int meuArray[] = {1, 2, 3, 4, 5};  
    int tamanho = sizeof(meuArray) / sizeof(meuArray[0]);  
    exibirArray(meuArray, tamanho);  
  
    return 0;  
}
```

```
struct Pessoa {  
    char nome[20];  
    int idade;  
};  
  
struct Pessoa criarPessoa() {  
    struct Pessoa pessoa;  
    strcpy(pessoa.nome, "João");  
    pessoa.idade = 25;  
    return pessoa;  
}  
  
int main() {  
    struct Pessoa pessoa1 = criarPessoa();  
  
    // Usar a struct retornada  
    printf("Nome: %s\n", pessoa1.nome);  
    printf("Idade: %d\n", pessoa1.idade);  
  
    return 0;  
}
```

Passagem de parâmetros por referência

Na passagem de parâmetros por **referência**, o endereço da variável é passado para a função. Logo, qualquer alteração feita na variável dentro da função **afetará o valor original** fora da função.

```
void modificarArray(int *array, int tamanho) {  
  
    // ...  
  
}  
  
int main() {  
  
    return 0;  
  
}
```

```
void modificarArray(int *array, int tamanho) {  
    // Modificar elementos do array  
    for (int i = 0; i < tamanho; i++) {  
        array[i] *= 2;  
    }  
}  
  
int main() {  
    int meuArray[] = {1, 2, 3, 4, 5};  
    int tamanho = sizeof(meuArray) / sizeof(meuArray[0]);  
  
    modificarArray(meuArray, tamanho);  
  
    // Exibir array modificado  
    for (int i = 0; i < tamanho; i++) {  
        printf("%d ", meuArray[i]);  
    }  
  
    return 0;  
}
```



```
struct Pessoa {  
    char nome[20];  
    int idade;  
};  
  
void modificarPessoa(struct Pessoa *p) {  
    p->idade += 1;  
}  
  
int main() {  
    struct Pessoa pessoa1 = {"João", 25};  
    modificarPessoa(&pessoa1);  
  
    // Exibir dados da pessoa  
  
    printf("Nome: %s\n", pessoa1.nome);  
    printf("Idade: %d\n", pessoa1.idade);  
  
    return 0;  
}
```

Retornos

Em C, para retornar uma variável composta em uma função, existem algumas opções, dependendo do tipo de variável composta que deseja retornar:

Retornando um array:

Uma opção é retornar um ponteiro para o primeiro elemento do array. No entanto, tenha cuidado ao retornar ponteiros para variáveis locais, pois elas deixarão de existir quando a função retornar.

```
int *criarArray(int tamanho) {  
    int *array = malloc(tamanho * sizeof(int));  
    // Preencher o array  
    for (int i = 0; i < tamanho; i++) array[i] = i + 1;  
    return array;  
}  
  
int main() {  
    int tamanho = 5;  
    int *meuArray = criarArray(tamanho);  
    // Usar o array retornado  
    for (int i = 0; i < tamanho; i++) printf("%d ", meuArray[i]);  
    free(meuArray); // Liberar a memória alocada  
    return 0;  
}
```

Retornando uma matriz:

O retorno é similar ao retorno de ponteiro para o primeiro elemento do array.

```
int **criarMatriz(int linhas, int colunas) {  
  
    int **matriz = malloc(linhas * sizeof(int *));  
  
    for (int i = 0; i < linhas; i++) {  
        matriz[i] = malloc(colunas * sizeof(int));  
        for (int j = 0; j < colunas; j++) {  
            matriz[i][j] = i + j;  
        }  
    }  
  
    return matriz;  
  
}
```



```
int main() {  
    int linhas = 3;  
    int colunas = 3;  
    int **minhaMatriz = criarMatriz(linhas, colunas);  
  
    // Usar a matriz retornada  
  
    for (int i = 0; i < linhas; i++) {  
        for (int j = 0; j < colunas; j++)  
            printf("%d ", minhaMatriz[i][j]);  
        printf("\n");  
    }  
  
    // Liberar a memória alocada  
  
    for (int i = 0; i < linhas; i++) {  
        free(minhaMatriz[i]);  
    }  
  
    free(minhaMatriz);  
    return 0;  
}
```



Q01

Implemente, em C, um algoritmo que declare um vetor (array) de inteiros com 5 elementos; preencha-o com valores digitados pelo usuário e, em seguida, exiba a soma de todos os elementos do array.

```
#include <stdio.h>

int main() {
    int vetor[5]; // Declarando um vetor de inteiros com 5 elementos
    int soma = 0; // Variável para armazenar a soma dos elementos

    printf("Digite 5 valores inteiros:\n");

    // Preenchendo o vetor com valores digitados pelo usuário
    for (int i = 0; i < 5; i++) {
        printf("Digite o valor %d: ", i + 1);
        scanf("%d", &vetor[i]);
        soma += vetor[i]; // Adicionando o valor ao somatório
    }

    // Exibindo a soma dos elementos do vetor
    printf("A soma dos elementos do vetor é: %d\n", soma);

    return 0;
}
```



Q02

Crie um programa que declare uma matriz de inteiros 3x3 e preencha-a com valores digitados pelo usuário. Em seguida, exiba a matriz na tela.

```
#include <stdio.h>

int main() {
    int matriz[3][3];
    printf("Digite os valores da matriz 3x3:\n");

    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            printf("Digite o valor da posição [%d][%d]: ", i, j);
            scanf("%d", &matriz[i][j]);
        }
    }
    printf("Matriz digitada:\n");
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            printf("%d ", matriz[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```



Q03

Crie uma struct chamada "Aluno" que armazene o nome e a nota de um aluno. Em seguida, declare um array de 3 elementos do tipo "Aluno" e preencha-o com dados digitados pelo usuário. Por fim, exiba os dados dos alunos na tela.

```
#include <stdio.h>

struct Aluno {
    char nome[50];
    float nota;
};

int main() {
    struct Aluno alunos[3];
    printf("Digite os dados dos alunos:\n");
    for (int i = 0; i < 3; i++) {
        printf("Aluno %d:\n", i + 1);
        printf("Nome: ");
        scanf("%s", alunos[i].nome);
        printf("Nota: ");
        scanf("%f", &alunos[i].nota);
    }
    printf("\nDados dos alunos:\n");
    for (int i = 0; i < 3; i++) {
        printf("Aluno %d:\n", i + 1);
        printf("Nome: %s\n", alunos[i].nome);
        printf("Nota: %.2f\n", alunos[i].nota);
    }
    return 0;
}
```




Q04

Crie um programa que declare um array de inteiros com 10 elementos e preencha-o com valores aleatórios. Em seguida, encontre e exiba o maior valor presente no array.

Handwriting practice area with 20 sets of three horizontal lines (top solid, middle dashed, bottom solid) for letter formation.



Q05

Crie um programa que declare uma matriz de inteiros 4x4 e preencha-a com valores aleatórios. Em seguida, encontre e exiba a soma de todos os elementos da matriz.

Handwriting practice area with 20 sets of three horizontal lines (top solid, middle dashed, bottom solid) for letter formation.



Q06

Crie uma struct chamada "Livro" que armazene o título, o autor e o ano de lançamento de um livro. Em seguida, declare um array de 5 elementos do tipo "Livro" e preencha-o com dados digitados pelo usuário. Por fim, exiba os dados dos livros na tela.

Handwriting practice area with 20 sets of three horizontal lines (top solid, middle dashed, bottom solid) for letter formation.



Q07

Crie um programa que declare um array de inteiros com 6 elementos e preencha-o com valores digitados pelo usuário. Em seguida, calcule e exiba a média dos valores presentes no array.

Handwriting practice area with 20 sets of three horizontal lines (top solid, middle dashed, bottom solid) for letter formation.



Q08

Crie um programa que declare uma matriz de inteiros 3x3 e preencha-a com valores digitados pelo usuário. Em seguida, encontre e exiba o maior valor presente na matriz.

Handwriting practice area with 20 sets of three horizontal lines (top solid, middle dashed, bottom solid) for letter formation.



Q09


Crie uma struct chamada "Funcionario" que armazene o nome, o salário e o cargo de um funcionário. Em seguida, declare um array de 4 elementos do tipo "Funcionario" e preencha-o com dados digitados pelo usuário. Por fim, exiba os dados dos funcionários na tela.

Handwriting practice area with 20 sets of three horizontal lines (top solid, middle dashed, bottom solid) for letter formation.



Q10

Crie um programa que declare um array de inteiros com 8 elementos e preencha-o com valores aleatórios. Em seguida, ordene os valores em ordem crescente e exiba o array ordenado.




Q11

Escreva uma função chamada "somaArray" que recebe um array de inteiros e seu tamanho como parâmetros e retorna a soma de todos os elementos do array.




Q12

Escreva uma função chamada "maiorValorMatriz" que recebe uma matriz de inteiros e suas dimensões como parâmetros e retorna o maior valor presente na matriz.




Q13

Escreva uma função chamada "concatenaStrings" que recebe duas strings e retorna uma nova string que é a concatenação das duas.




Q14

Escreva uma função chamada "mediaValores" que recebe um array de números reais e seu tamanho como parâmetros e retorna a média dos valores.



Q15

Escreva uma função chamada "somaElementosMatriz" que recebe uma matriz de inteiros e suas dimensões como parâmetros e retorna a soma de todos os elementos da matriz.



Q16

Escreva uma função chamada "maiorIdade" que recebe um array de structs "Pessoa" contendo o nome e a idade de pessoas, e seu tamanho como parâmetros, e retorna a idade da pessoa mais velha.




Q17

Escreva uma função chamada "multiplicaMatriz" que recebe duas matrizes de inteiros e suas dimensões como parâmetros e retorna uma nova matriz que é o resultado da multiplicação das duas.



Q18

Escreva uma função chamada "buscaElementoArray" que recebe um array de inteiros, seu tamanho e um número como parâmetros, e retorna verdadeiro se o número estiver presente no array, ou falso caso contrário.



Q19

Escreva uma função chamada "ordenaArray" que recebe um array de inteiros e seu tamanho como parâmetros e retorna o array ordenado em ordem crescente.



Q20

Escreva uma função chamada "somaValoresStruct" que recebe um array de structs "Valores" contendo dois inteiros, e seu tamanho como parâmetros, e retorna a soma dos valores de cada struct.